

Models for Sequential Data

Machine Learning 2019
mlvu.github.io

the plan

part 1:

Sequences

Markov models

Embedding models

Word2Vec

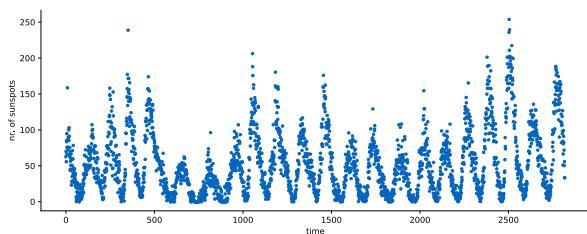
part 2:

Recurrent Neural Nets

LSTMs

2

numeric 1-dimensional



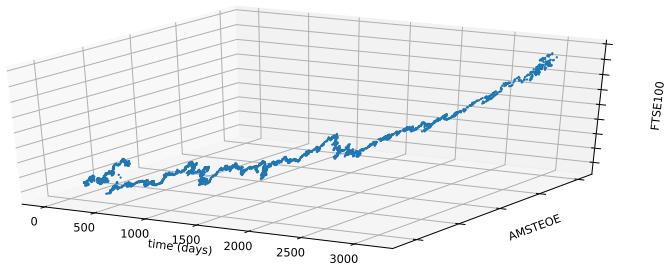
Before we start looking at different models to learn from sequential data, it pays to think a little bit about the different types of sequential datasets we might encounter.

As with the traditional setting (a table of independently sampled instances) we can divide our features into numeric and discrete.

A single 1D sequence might look like this. We could think of a stock price over time, traffic to a webserver, or atmospheric pressure over Amsterdam. In this case, the data shows the number of sunspots observed over time.

3

numeric n-dimensional



4

Sequential numeric data can also be multidimensional. In this case, we see the closing index of the AEX and the FTSE100 over time. This data is a sequence of 2D vectors.

symbolic (categorical) 1-dimensional

the, cat, sat, on, the, mat

t, h, e, _, c, a, t, _, s, a, t, _, o, n, _, t, h, e, _, m, a, t

5

If the elements of our data are discrete (analogous to a categorical feature), it becomes a sequence of symbols. Language is a prime example. In fact, we can model language as a sequence in two different ways: as a sequence of words, or as a sequence of characters.

symbolic n-dimensional

the/ART cat/NOUN sat/VERB on/PREP
the/ART mat/NOUN

We can also encounter sequences with multiple categorical features per timestamp, like music, or tagged language. These can often be difficult to represent.



6

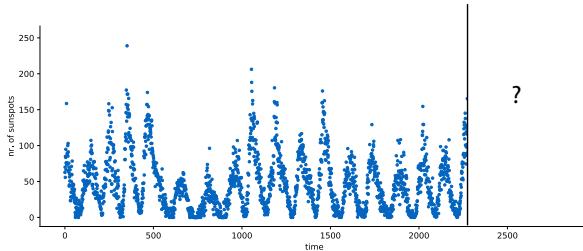
single sequence vs set-of-sequences

Dear recipient, Avangar Technologies announces the beginning of a new unprecedented global employment campaign ...	spam
BENCHMARK SUPPLY 7540 BRIDGEGATE COURT ATLANTA GA 30350 ***LASER PRINTER TONER CARTRIDGES*** ***FAX AND COPIER TONER***	spam
On 08/08/02 15:26 -0700, Chip Paswater wrote: J > Well, a little more than one bit -- you have to transmit the signature, plus now the entire ...	ham
If you own a travel related website, why not submit your site to our directory. Just select the appropriate category and subcategory and enter your title ...	spam
hmm, I assume you're going to report this to the nmh folks? Yes, I will, sometime, after I look at the nmh sources and see what they have managed to break, ...	ham
Hello I am emailing you as we found your holiday property at [Source] and would love you to list it on our website for free:	spam
Hello again self-catering.co.uk was recently launched	spam

Then there is the question of what we're trying to predict. Do we have one sequence per instance, and are trying to, for instance, classify the sequences? That's what we see here: each email is a sequence (of words or characters).

The instances themselves do not have a very sequential ordering (emails do have a timestamp, but this ordering is usually ignored).

single sequence



An entirely different setting is one where the dataset as a whole is a sequence, and the instances are ordered.

In that case, we often want to predict the future values of the sequence based on what we've seen in the past. To keep things simple, let's stick with 1D numeric data, like our sunspots example.

single sequence: feature extraction

1 3 5 3 4 2 3 4 5 5 4 3 2 1 3 2 2

t-3	t-2	t-1	t
5	2	3	3
3	5	2	3
3	3	5	2
5	3	3	5
4	5	3	3
2	4	5	3
3	2	4	5
3	3	2	4
1	1	3	2
2	3	3	3

One simple way to achieve this, is to translate it to a classic regression problem by representing each point by the m values before it. This gives us a table with a target label (the value at time t) and m features (the m preceding values). If we successfully train a regression model on this task, we can then use it to predict future values of the sequence.

Many other features are possible: mean over the whole history. Mean over the last 10 points, variance over the last 10 points, etc.

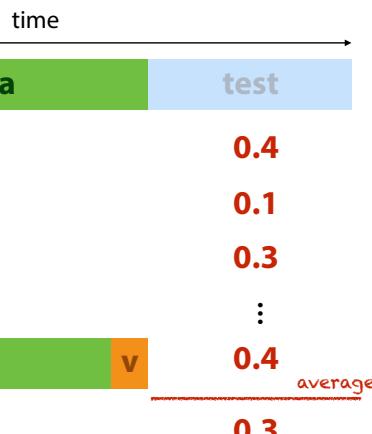
But. There's a problem. If we shuffle this data, and pick a test and training set, the classifier is trained on data that happens in the future to data it will be tested on!

Think about the REAL-WORLD use case.

Whenever you have questions about how to approach something like this, it's best to think about the real world setting where you might apply your trained model.

In this case, we are going to apply the prediction model on the three most recent values, to predict the next value. The test data should simulate that situation. That means that the training data should not contain instances that occur later than the test data.

walk-forward validation



To check your algorithm, to do cross validation.

testing

time →

training data test

train

train

train

⋮

train

Note, only if the target labels have a meaningful ordering in time. In spam classification, for instance, this is not necessary (emails do have a timestamp, but seeing an email from the future does not usually give you an unrealistic advantage).

summary: sequential data

Sequences: consisting of numbers or symbols

Both is pretty rare

Dataset: consisting of a **sequence per instance**, or a **sequence of instances**.

For a sequence of instances, careful with your test and validation

Both can be converted to fit classic machine learning through feature extraction.

Rest of the lecture: 1D symbolic sequences.

Extension to nD and/or numeric is often trivial

But, often it's better to take a model that can consume sequences natively.

13

modeling sequences: probability

$$\begin{aligned} p(\text{"congratulations you have won a prize"}) \\ = p(W_1=\text{congratulations}, W_2=\text{you}, W_3=\text{have}, \\ W_4=\text{won}, W_5=\text{a}, W_6=\text{prize}) \end{aligned}$$

$$p(W_1, W_2, W_3, W_4, W_5, W_6)$$

14

We'll start by looking at some ways to model sequences in probabilistic terms.

When modelling probability, we usually break the sequence up into its **tokens** (in this case the words of the sentence) and model each as a random variable. Note that these random variables are decidedly *not* independent.

This leaves us with a joint distribution over 6 variables, which we would somehow like to model and fit to a dataset.

slide 22, Probabilistic Methods 1

$$p(x, y) = p(x | y)p(y)$$

Remember this rule from the first Probabilistic Methods lecture. If we have a joint distribution over more than two variables on the left, we can apply this rule multiple times.

chain rule of probability

$$p(W_4, W_3, W_2, W_1)$$

$$= p(W_4, W_3, W_2 | W_1)p(W_1)$$

$$= p(W_4, W_3 | W_2, W_1)p(W_2 | W_1)p(W_1)$$

$$= p(W_4 | W_3, W_2, W_1)p(W_3 | W_2, W_1)p(W_2 | W_1)p(W_1)$$

$$p(\text{prize} | \text{a, won, have, you, congratulations})$$

This gives us the **chain rule of probability** (not to be confused with the chain rule of calculus, which is entirely different), which is often used in modelling sequences.

The chain rule allows us to break a joint distribution on many variables into a product of conditional distributions. In sequences, we often apply it so that each word becomes conditioned on the words before it.

$$\log p(\text{sentence}) = \sum_{\text{word}} \log p(\text{word} | \text{words before word})$$

In other words, we can rewrite the probability of a sentence as the product of the probability of each word, conditioned on its history. If we use the log probability, this becomes a sum.

Note that applying the chain rule in a different order would allow us to condition any word on any other word, but conditioning on the history fits well with the sequential nature of the data, and will allow us to make some useful simplifications later.

$$p(W | \text{the, man, fell, out, of, the})$$

the man fell out of the ...

window

aquarium

pool

cycling

A perfect language model would encompass everything we know about language: the grammar, the idiom and the physical reality it describes. For instance, it would give *window* a very high probability, since that is a very reasonable way to complete the sentence. *Aquarium* is less likely, but still physically possible and grammatically correct. A very clever language model might know that falling out of a *pool* is not physically possible (except under unusual circumstances), so that should get a lower probability, and finally *cycling* is ungrammatical, so that should get very low probability (perhaps even zero).

The problem is most sentences of this length will never have been seen before in their entirety. In order to simplify the model, we simply limit its memory.

Markov assumption

$$p(\text{prize} | \text{a, won, have, you, congratulations}) = p(\text{prize} | \text{a, won})$$

This is called a **Markov assumption**. It's a bit like the naive Bayes assumption: we know it's incorrect, but it still yields a very usable model. The number of words we retain in the conditional is called the order of the Markov model: this is a Markov assumption for a second-order Markov model.

19

$$\begin{aligned} p(\text{prize}, \text{a, won, have, you, congratulations}) \\ = & p(\text{prize} | \text{a, won}) p(\text{a} | \text{won, have}) p(\text{won} | \text{have, you}) \\ & p(\text{have} | \text{you, congratulations}) p(\text{you} | \text{congratulations}) p(\text{congratulations}) \end{aligned}$$

$$p(\text{prize} | \text{a, won}) \approx \frac{\# \text{"won a prize"}}{\# \text{"won a"}}$$

20

Using the Markov assumption and the chain rule together, we can model a sequence as limited-memory conditional probabilities. These probabilities can then be very simply estimated from a large dataset of text (called a **corpus**).

To estimate the probability of **prize** given "won a" we just count how often "won a prize" occurs as a proportion of the times "won a" occurs. In other words, how often "won a" is followed by **prize**.

These n-word snippets are called **n-grams**. "won a prize" is a **trigram**, and "won a" is a **bigram**.

sequential sampling from a language model

start with a small *seed* sequence $s = [w_1, w_2, w_3]$ of tokens.

loop:

Sample next word w according to $p(W = w | w_1, w_2, \dots)$

append w to s

21

One interesting thing we can do with a markov model, is to sample from it, step by step. We start with a seed of a few words, and then work out the probability distribution over the next word, given the last n words. We sample from this distribution, append the sample to our text and repeat the process.

Note that with the Markov assumption, we only need the last n elements of the sequence to work out the probabilities.

sequential sampling: Markov chain

Go thy

ways, wench; serve them, joining their best friend to ransom straight,
And make him dead, deceased, she's dead, she's good, thou hast thou dost thou
dost excuse.

Is thy son to church?

CAPULET Ready to love now

Doth grace that I beseech your high majesty.

SIR WALTER BLUNT, with tears: mine ear. Prithee,
tell her you both of Rome of your will: I
pray you, daughter, he is my soul, he proclaim'd
By Richard that thou dead:
Then, as herbs, grace himself an hour.

source <http://www.schmipsum.com/> 22

Here is a bit of text sampled from a Markov model trained on the works of Shakespeare.

(We're essentially training a very simple probabilistic predictor in the fashion of slide 9, and then sampling from it sequentially)

sequence classification

$p(\text{spam} | \text{"congratulations, you have won a prize"})$

Let's now look at a sequence-per-instance example: the spam classification task we saw earlier. We'll see how to approach this with a Markov model.

We are after the probability of the class, given the contents of the message.

Bayes classifier

$p(\text{spam} | W_1, \dots, W_n) \propto p(W_1, \dots, W_n | \text{spam})p(\text{spam})$

We'll train a generative/Bayes classifier and use Bayes rule to flip around the probabilities.

$p(\text{spam})$ can be estimated as the proportion of spam emails in our data set. For the probability of the message given the class, we'll use our language model.

24

conditional on class

```
p(prize, a, won, have, you, congratulations | spam)
= p(prize | a, won, spam) p(a | won, have, spam)
p(won | have, you, spam) p(have | you, congratulations, spam)
p(you | congratulations, spam) p(congratulations, spam)
```

$$p(\text{prize} | \text{a}, \text{won}, \text{spam}) \approx \frac{\#\text{spam}^{\text{"won a prize"}}}{\#\text{spam}^{\text{"won a"}}$$

We use the chain rule and the Markov assumption to define the probability that a message occurs (given that it's **spam**).

We then estimate the different conditional probabilities by computing the relative frequencies of bigrams and trigrams, as before, but we compute them only over the **spam** part of our data.

25

algorithm: 3rd-order markov model

training:

```
for each class c:
    for n in 1, 2, 3:
        count n-grams in all text belonging to class c
```

classification:

```
given text w1, ..., wm
log p(w1, ..., wm | c) =
    log p(w1 | c) + log p(w2 | w1, c) +
    ∑i ∈ [3, ..., m] log p(wi | wi-1, wi-2, c)
```

26

Markov modeling: final comments

- **0-order Markov model: Naive Bayes**
For spam classification higher orders don't improve performance. For other tasks they do.
- **Short documents are vastly more likely than long ones**
Doesn't matter for classification. In other settings, conditioning on length may be necessary.
- **Laplace smoothing**
Same as before: adapt the estimator by adding pseudo-observations

V is the vocabulary (the total number of words known to the language model)

$$p(\text{prize} | \text{a}, \text{won}) \approx \frac{1 + \#\text{"won a prize"}}{|V|^3 + \#\text{"won a"}}$$

27

the man fell out of the ...
pool
gazebo
pool

"congratulations you have won a prize"

$p(\text{spam}|\text{"congrats, a reward has been awarded to you"})$

These kinds of probability models, and many other models for symbolic sequential data, is that they model symbols as entirely separate objects. They don't exploit the fact that some words have similar meanings of syntactic functions.

For instance, the Markov assumption helps us with the man falling out of the pool, because "out of the pool" is a common phrase, even if nobody falls out of it. However, for rarer cases like "out of the gazebo" vs "out of the cycling" it may be that both trigrams never appear in the data. In that case, we might still say that the first is more likely because it at least ends in a *noun*, which is expected.

Similarly, if our training data give us little information about the message shown here, we would want a model that can conclude that the words in the sentences are very similar to words that do occur in the spam training corpus.

28

embedding models

Model object x by embedding vector e_x .

If e_x and e_y are "similar," so are x and y

Learn the parameters of $\{e_x\}$ from data.

Cf: latent vectors from autoencoders, PCA, etc.

One way to solve this problem this is to see symbols in our sequence not just as atomic objects, but to represent them as vectors, and to let the similarities of these vectors represent the similarities between the words.

(This is a bit like mapping images to latent vectors in an autoencoder.)

29

distributional hypothesis

Words that occur in the same context often have similar meanings.

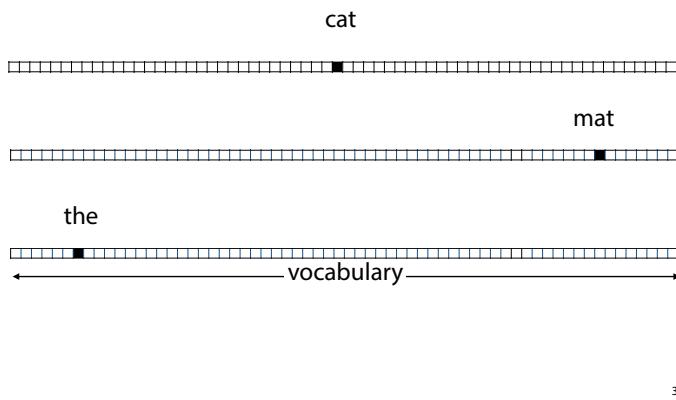
The **distributional hypothesis** provides us with a sound way to train these embeddings.

To paraphrase: if we have some word X, and we create a probability distribution P modelling how likely a word Y is to occur in the context of X, we can use that distribution P as a proxy for the *meaning* of X.

The algorithm we will discuss, Word2Vec, provides learns a simple mapping from the embedding to a probability distribution.

30

1-hot vector



To start with, we'll represent words as atomic objects: in a single, very large one-hot vector.

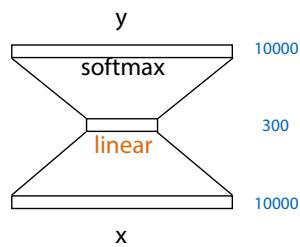
31

Word2Vec (skipgram version)

shall i compare thee to a summers day thou art more lovely ...

x

x	y
compare	shall
compare	i
compare	thee
compare	to
thee	i
thee	compare
thee	to
thee	a
to	compare
to	thee
to	a
to	summers
a	thee
a	to
a	summers
a	day
summers	to



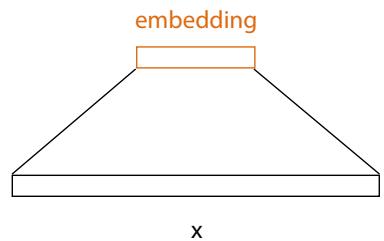
32

We slide a context window over the sequence. The task is to predict the distribution $p(y|x)$: that predict which words are likely to occur in the context window given the middle word.

We create a dataset of word pairs from the entire text and feed this to a very simple two-layer network. This is a bit like an autoencoder, except we're not reconstructing the output, but predicting the *context*.

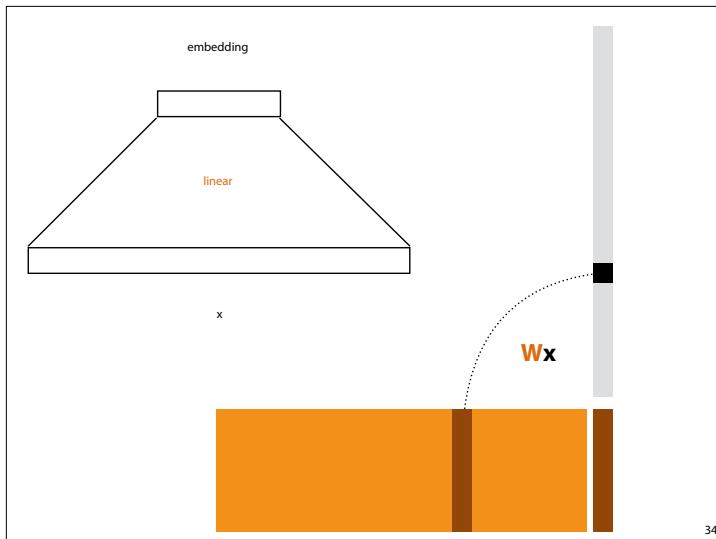
The softmax activation over 10k outputs is very expensive to compute, and you need some clever tricks to make this feasible (called *hierarchical softmax* or negative sampling). We won't go into them here.

after training



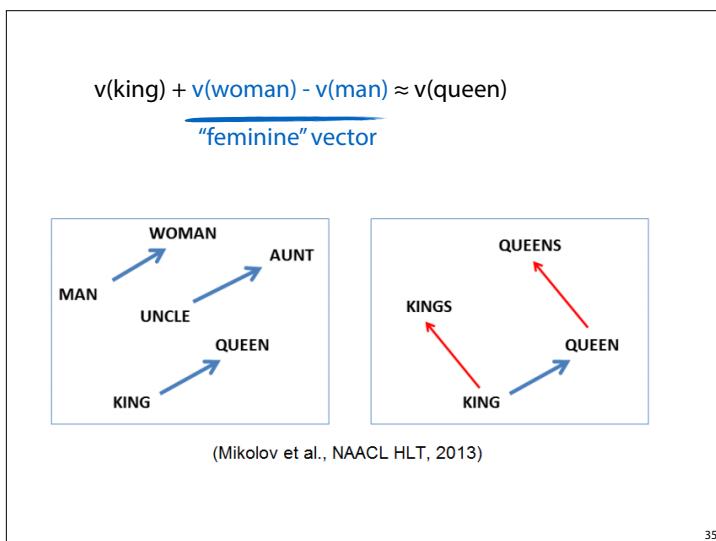
After training, we discard the second layer, and use only the embeddings produced by the first layer.

33

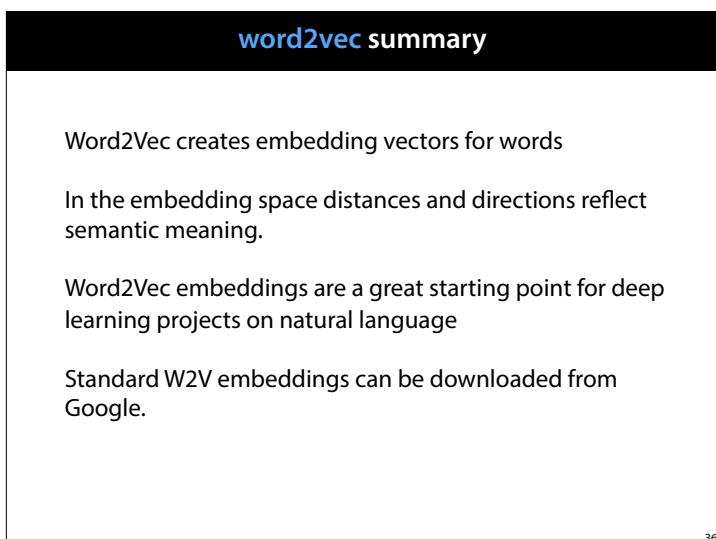


Because the input layer is just a matrix multiplication, and the input is just a one-hot vector, what we end up doing when we compute the embedding for word i , is just extracting the i -th column from W .

In other words, we're not really training a function that computes an embedding for each word, we are actually learning the embeddings directly: every element of every embedding vector is a separate parameter.



Famously, Word2Vec produces not just an informative embedding, where similar words are close together, but for many concepts, there seems to be a kind of algebraic structure, similar to the smile vector example from the autoencoder.



break

Garkov



by Josh Millard
via Jim Davis

9-29

Garkov



by Josh Millard
via Jim Davis

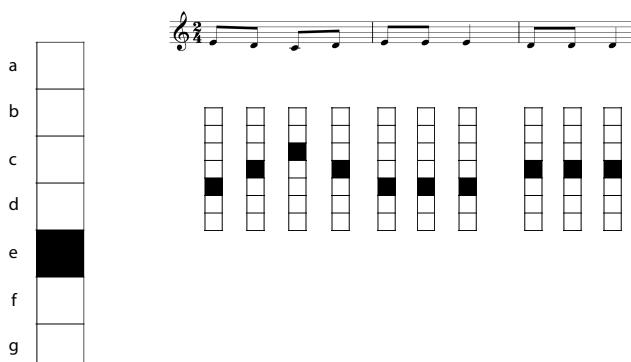
7-26

Garfield generated by a Markov model

source: <https://blog.codinghorror.com/markov-and-you/>

37

sequences in deep learning



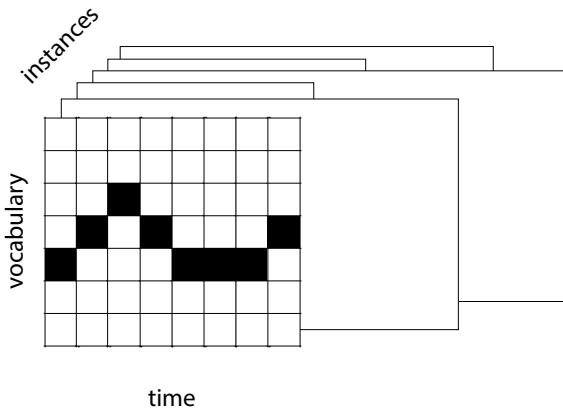
38

As we've seen, when we want to do deep learning, our input should be represented as a tensor. Preferably in a way that retains all information (i.e. we want to be learning from the raw data, or something as close to it as possible).

Here is an example: to encode a simple monophonic musical sequence, we just one-hot encode the notes, and encode note sequence as a matrix: one dimension for time, one dimension for the notes.

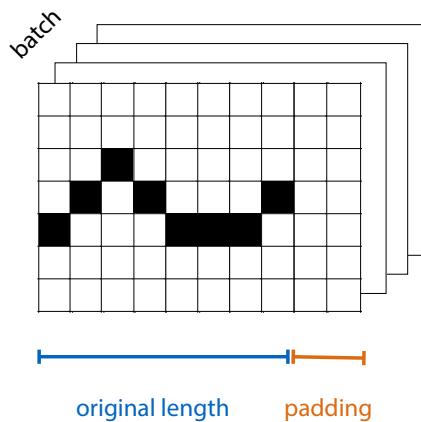
source: <https://violin-sheetmusic.org>

If we have multiple sequences, this leads to a data set of



39

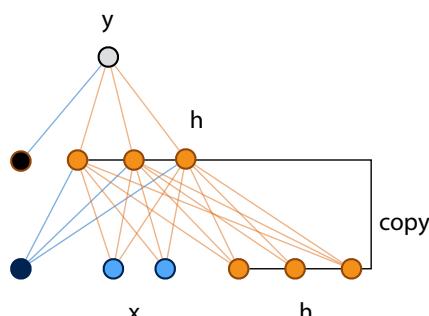
batching sequences



In principle, this is not a problem, we want to build models that can deal with sequences of any length (and can generalise over sequences of variable lengths).

However, within a batch it's usually required that all sequences have the same length. Often, we put sequences of similar length together (for instance by sorting the data by length) and then pad the shorter sequences with zero vectors, so that all sequences are the same length.

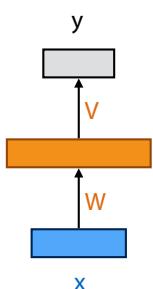
recurrent neural network



One of the most popular neural networks for sequences is the **recurrent neural network**. This is a generic name for any neural network with cycles in it.

The figure shows a popular configuration. It's a basic fully connected network, except that its input x is extended by three nodes to which the hidden layer is copied.

visual shorthand

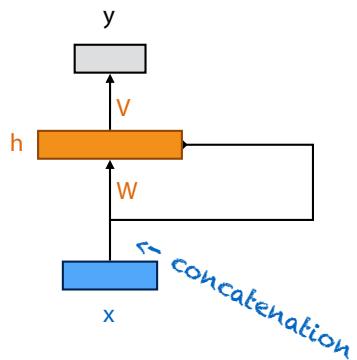


To keep things clear we will adopt this visual shorthand: a rectangle represents a vector of nodes, and an arrow feeding into such a rectangle annotated with a weight matrix represents a fully connected transformation.

We will assume bias nodes are included.

Adapted from
<http://colah.github.io/posts/2015-08-Understanding-LSTMs/>

visual shorthand

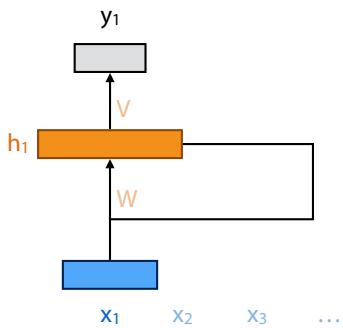


A line with no weight matrix represents a copy of the input vector. When two lines flow into each other, we concatenate their vectors.

Here, the added line copies h , concatenates it to x , and applies weight matrix W .

43

RNNs on sequences

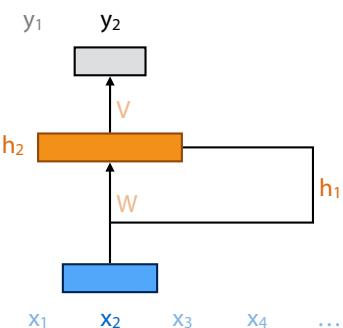


We can now apply this neural network to a sequence. We feed it the first input, x_1 , result in a first value for the hidden layer, h_1 , and retrieve the first output y_1 .

The hidden nodes are initialise to zero, so at first the network behaves just like a fully connected network.

44

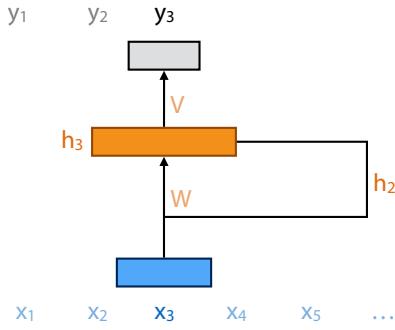
RNNs on sequences



We then feed the second input in the sequence, x_2 . We now receive the *previous* hidden layer, h_1 , concatenate it to the input, and multiply it by W , to produce the second hidden layer h_2 . This is multiplied by V to produce the second output.

45

RNNs on sequences



And so on.

46

how to train RNNs?

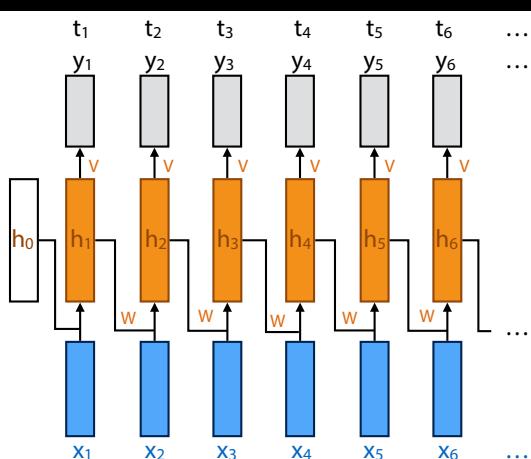
Provide an input sequence \mathbf{x} and a target sequence \mathbf{t} .

Backpropagation Through Time (BPTT).

In principle, we

47

how to train RNNs? unrolling



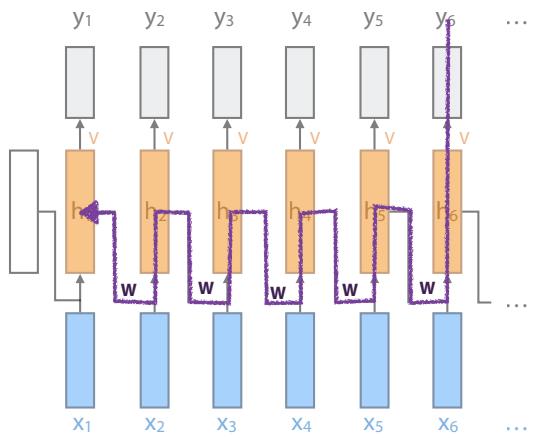
Instead of visualising a single small network, applied at every time step, we can **unroll** the network. Every step in the sequence is applied in parallel to a copy of the network, and the recurrent connection flows from the previous copy to the next.

Now the whole network is just one big, complicated **feedforward net**, that is, a network *without* cycles. Note that we have a lot of shared weights, but we know how to deal with those.

The hidden layer is initialised to the zero vector.

48

backpropagation through time



Now the whole network is just one big, complicated feedforward net. Note that we have a lot of shared weights, but we know how to deal with those.

what can we use RNNs for?

Sequence to sequence:

Eg. Translating English to French

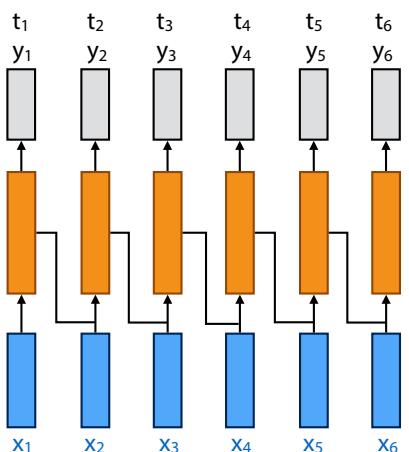
Sequence to label

Eg. sequence classification, regression

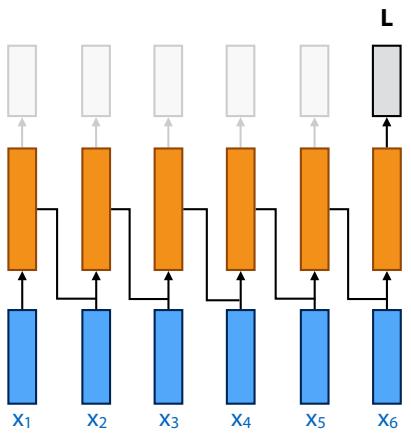
Label to sequence

Eg. (conditional) sentence generation

sequence to sequence



sequence to label

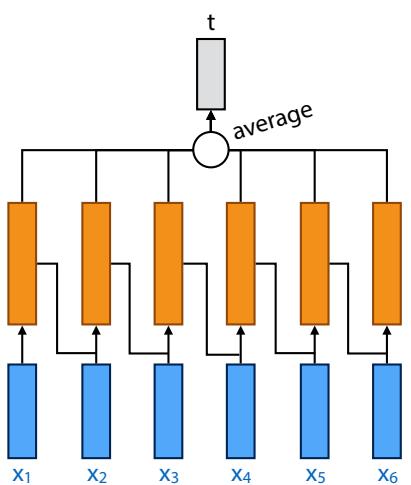


If we want to map sequences to labels, we could just ignore all outputs except the last, and take that to be the label. We can compare it to the target label and back propagate the error.

However, this would make it much

52

sequence to label

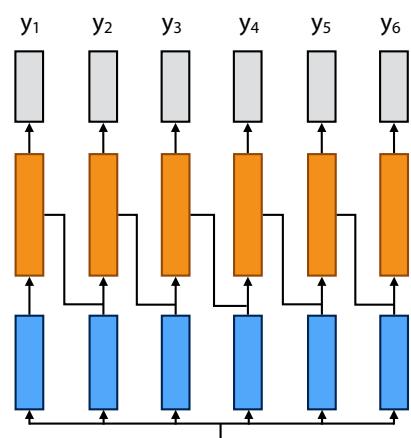


A better approach is to average the outputs over the whole sequence. This makes the distance to the label the same for every input.

Note that the model should be able to handle sequences of variable lengths: this means we can't put a fully connected layer on the top to combine the whole sequence into a line label.

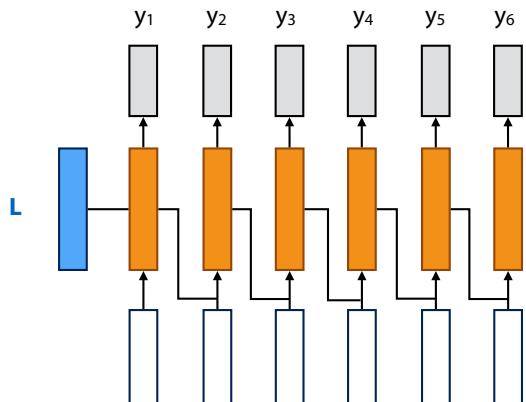
53

label to sequence



54

label to sequence



We can also use the hidden layer at timestep 0 as an input for the label, and zero the sequence inputs (or use them for something else).

55

the problem of long-term dependence

I was born in France, as matter of fact in a little village near Paris, it's famous for its pain-au-chocolat, I lived there until I was 16, when I moved to Amsterdam, so I'm fluent in...

French
Dutch
Aquarium

Basic RNNs work pretty well, but they do not learn to remember information for very long. Technically they can, but the gradient vanished too quickly over the timesteps.

You can't have a long term memory for everything. You need to be selective, and you need to learn to select words to be stored for the long term when you first see them. In order to remember things long term you need to forget many other things.

56

LSTM (1997)

Long short-term memory

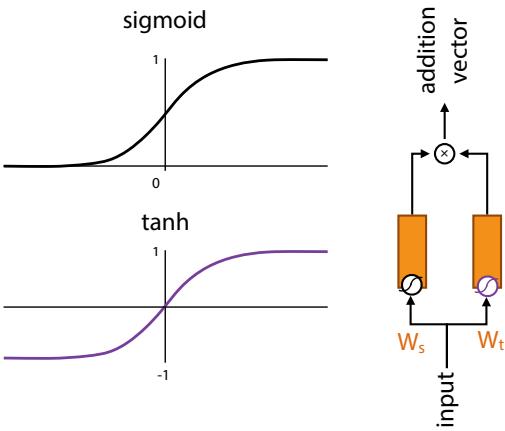
Selective forgetting and remembering, controlled by learnable "gates"

Possibly the first successful deep neural network

An enduring solution to the problem are LSTMs. LSTMs have a complex mechanism, which we'll go through step by step, but the main component is a gating mechanism.

57

gate

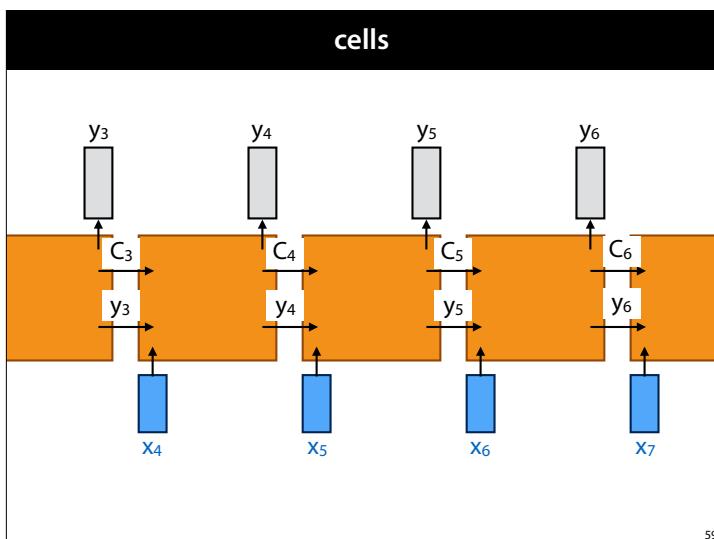


The gate combines the sigmoid and tanh activations. The sigmoid we've seen already. The tanh is just a the sigmoid rescaled so that its outputs are between -1 and 1.

The gating mechanism takes two input vectors, and combines them using a sigmoid and a tanh activation. The gate is best understand as producing an additive value: we want to figure out how much of the input to add to some other vector (if it's import, we want to add most of if, otherwise, we want to forget it, and keep the original value).

The input is first transformed by two weight metrics and then passed though a sigmoid and a tanh. The tanh should be thought of as a mapping of the input to the range [-1, 1]. This ensures that the effect of the addition vector can't be too much. The sigmoid acts as a selection vector. For elements of the input that are important, it outputs 1, retaining all the input in the addition vector. For elements of the input that are not important, it outputs 0, so that they are zeroed out. The sigmoid and tanh vectors are element-wise multiplied.

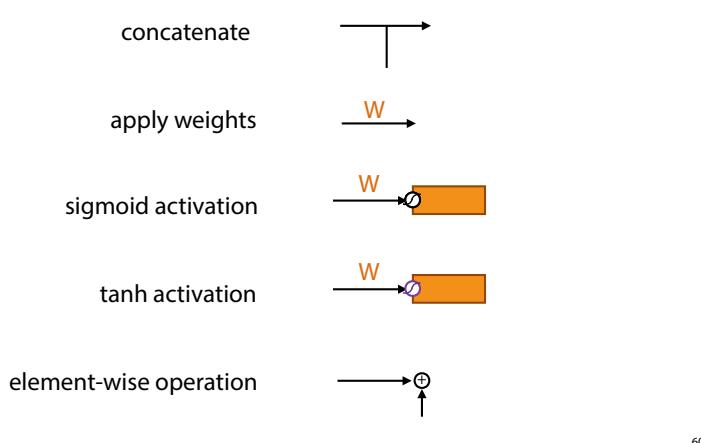
cells

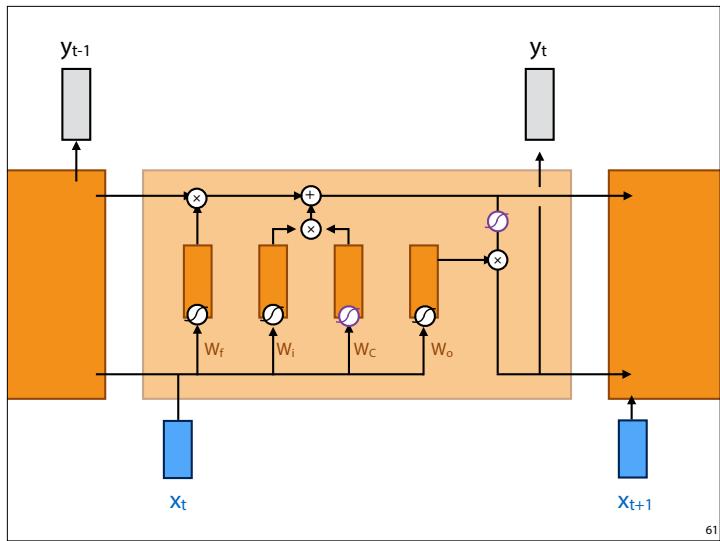


The basic operation of the LSTM is called a cell (the orange square, which we'll detail later). Between cells, there are two recurrent connections, y , the current output, and C the **cell state**.

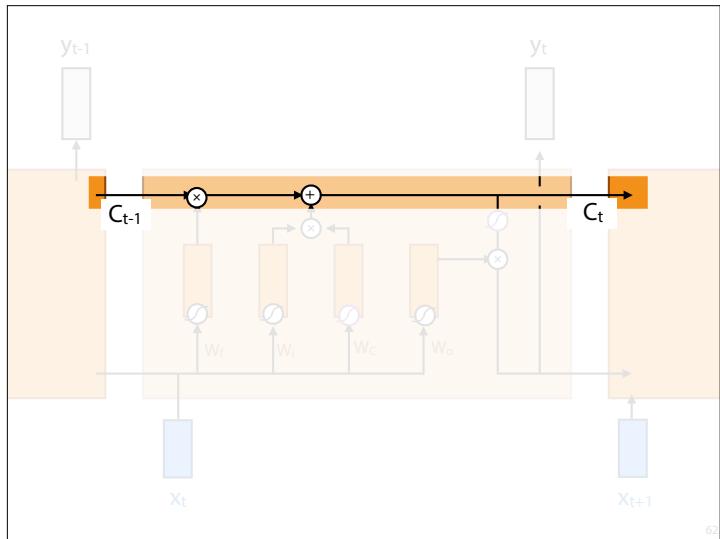
notation

Here is our visual notation.



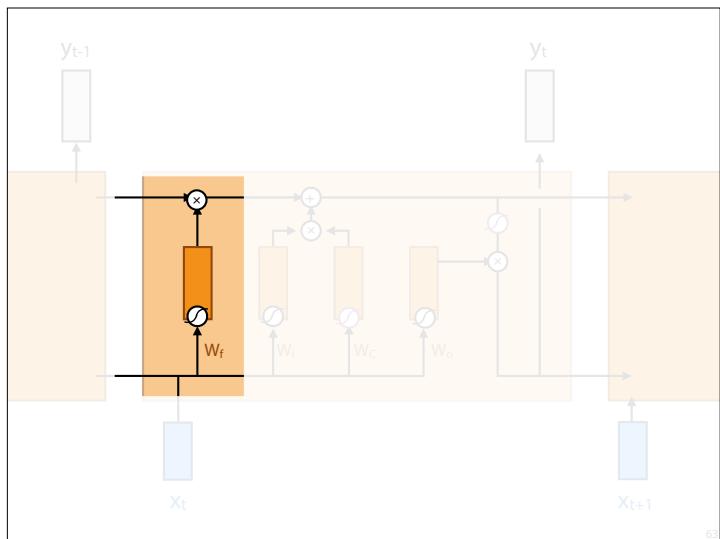


Here is what happens inside the cell. It looks complicated, but we'll go through all the elements step by step.



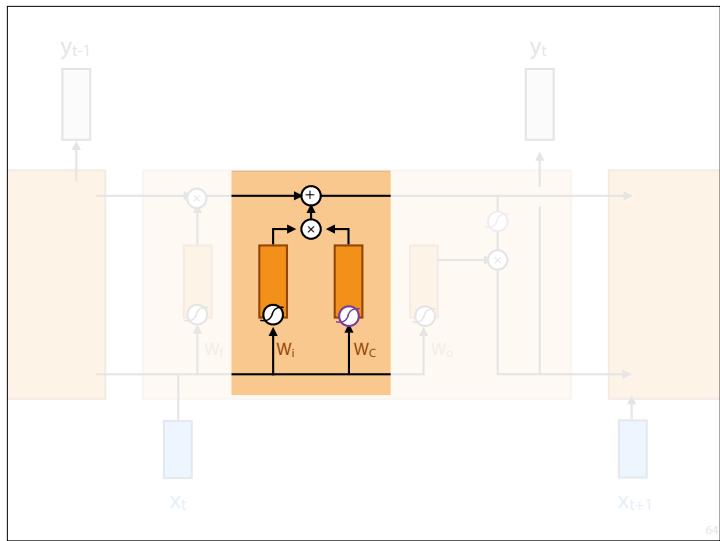
The first is the “conveyor belt”. It passes the previous cell state to the next cell. Along the way, the current input can be used to manipulate it.

Note that the connection from the previous cell to the next has *no activations*. This means that along this path, gradients do not decay. It's also very easy for an LSTM cell to ignore the current information and just pass the information along the conveyor belt.

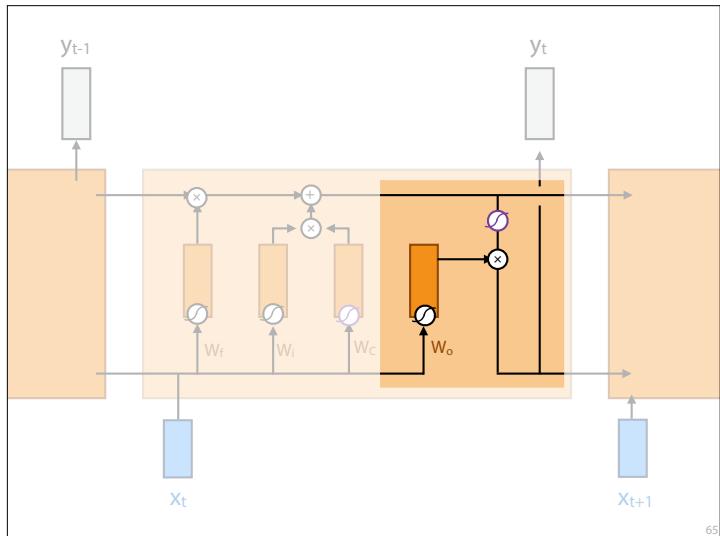


Here is the first manipulation of the conveyor belt. This is called the **forget gate**.

It looks at the current **input**, concatenated with the previous **output**, and applies an element-wise scaling to the current value in the conveyor belt. Outputting all 1's will keep the current value on the belt what it is, and outputting all values near 0, will decay the values (forgetting what we've seen so far, and allowing it to be replaced by our new values in the next step).



in the next step, we pass the input through a generic gate, as described earlier, and add the resulting vector to the value on the conveyor belt.



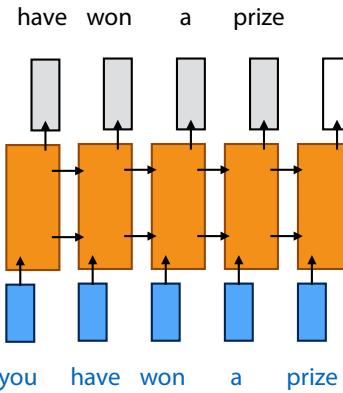
Finally, we need to decide what to output now. We take the current value of the conveyor belt, tanh it to rescale, and element-wise multiply it by another sigmoid activated layer. This layer is sent out as the current output, and sent to the next cell along the second recurrent connection.

LSTM language model

$$p(\text{prize} \mid \text{a, won, have, you, congratulations})$$

That's a lot of complexity. Let's see what it buys us. The best way, by far, to illustrate the power of LSTMs is to apply the sequential sampling trick.

LSTM language model

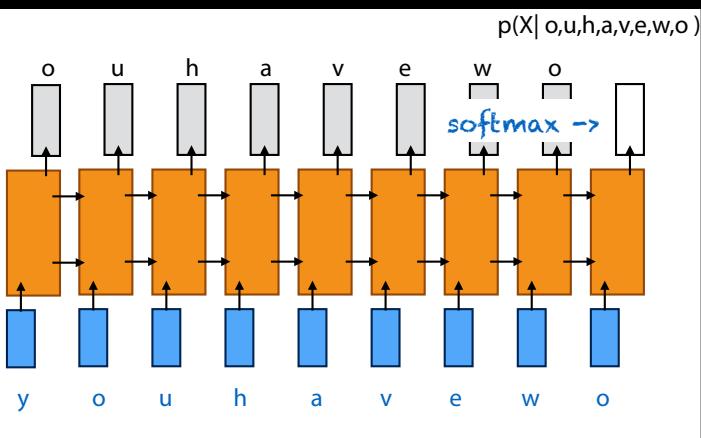


We train the language model to predict the next token in the sequence. This is easy to do: we just use the sequence shifted by 1 as the training sequence.

67

character level language model

We'll do this at character level, to make it more challenging, and we ensure that the network outputs a softmax activated vector over all characters at each timestep. We can then interpret the output of the model at step t as the probability of the character at that point, conditioned on the entire history.



68

sequential sampling from a language model

start with a small seed sequence $s = [c_1, c_2, c_3]$ of tokens.

loop:

Sample next char c according to $p(C = c | c_1, c_2, \dots)$

feed the whole seed to the network

append c to s

Note that this time, there is no Markov assumption. The network has to see the whole sequence so far to predict the next character.

also known as a *autoregressive RNN*

69

some examples

source: The Unreasonable Effectiveness of Recurrent Neural Networks

Andrej Karpathy

<http://karpathy.github.io/2015/05/21/rnn-effectiveness/>

70

Shakespeare

PANDARUS:

Alas, I think he shall be come
approached and the day
When little strain would be
attain'd into being never fed,
And who is but a chain and
subjects of his death,
I should not sleep.

Second Senator:

They are away this miseries,
produced upon my soul,
Breaking and strongly should be
buried, when I perish
The earth and thoughts of many

Remember, this is a **character level** language model.

71

Wikipedia

Naturalism and decision for the majority of Arab countries' capitalide was grounded by the Irish language by [[John Clair]], [[An Imperial Japanese Revolt]], associated with Guangzham's sovereignty. His generals were the powerful ruler of the Portugal in the [[Protestant Immineners]], which could be said to be directly in Cantonese Communication, which followed a ceremony and set inspired prison, training. The emperor travelled back to [[Antioch, Perth, October 25|21]] to note, the Kingdom of Costa Rica, unsuccessful fashioned the [[Thrales]], [[Cynth's Dajoard]], known in western [[Scotland]], near Italy to the conquest of India with the conflict. Copyright was the succession of independence in the slop of Syrian influence that was a famous German movement based on a more

72

of Costa Rica, unsuccessful fashioned the [[Thrales]], [[Cynth's Dajoard]], known in western [[Scotland]], near Italy to the conquest of India with the conflict. Copyright was the succession of independence in the sloop of Syrian influence that was a famous German movement based on a more popular servitious, non-doctrinal and sexual power post. Many governments recognize the military housing of the [[Civil Liberalization and Infantry Resolution 265 National Party in Hungary]], that is sympathetic to be to the [[Punjab Resolution]] (PJS)[<http://www.humah.yahoo.com/guardian.cfm/7754800786d17551963s89.htm>] Official economics Adjoint for the Nazism, Montgomery was swear to advance to the resources for those Socialism's rule, was starting to signing a major tripad of aid exile.]]

73

```
<page>
  <title>Antichrist</title>
  <id>865</id>
  <revision>
    <id>15900676</id>
    <timestramp>2002-08-03T18:14:12Z</timestramp>
    <contributor>
      <username>Paris</username>
      <id>23</id>
    </contributor>
    <minor />
    <comment>Automated conversion</comment>
    <text xml:space="preserve">#REDIRECT
[[Christianity]]</text>
  </revision>
</page>
```

74

LaTeX

For $\bigoplus_{i=1}^n \mathcal{F}_i$, where $\mathcal{L}_{ii} = 0$, hence we can find a closed subset H in \mathcal{H} and any sets \mathcal{F}_i on X , U is a closed immersion of S , then $U \rightarrow T$ is a separated algebraic space. Proof. Proof of (1). It also start we get $S = \text{Spec}(R) = U \times_X U \times_X U$ and the compatibility in the fibre product we have to prove the lemma generated by $\prod_i Z_i$ for $i \in I$. Consider the maps U along the set of points S_{affine} , and $U \rightarrow U'$ is the fiber category over S in U in Section ?? and the fact that any U' affine, see Morphisms, Lemma ?? Hence we obtain a scheme S and any open subset $W \subset U$ in $\text{Sh}(G)$ such that $\text{Spec}(R') \rightarrow S$ is smooth or an $U = \bigcup_i U_i \times_S U_i$

To prove study we see that $\mathcal{F}|_U$ is a covering of X' , and $T|_U$ is an object of $\mathcal{F}|_U$ for $i > 0$ and \mathcal{F}_p exists and let \mathcal{F}_p is a presheaf of \mathcal{O}_X -modules on C as a \mathcal{F} -module. In particular $\mathcal{F} = U/\mathcal{F}$ we have to show that

$\tilde{\mathcal{M}} = \mathcal{I}^* \otimes_{\mathcal{O}(S)} \mathcal{O}_{X,p} = i_2^* \mathcal{F}$ is a unique morphism of algebraic stacks. Note that $\text{Arrows} = (\text{Sch}/S)_{fppf}^\text{opp}, (\text{Sch}/S)_{fppf}$ and

$V = U/S, \mathcal{O} \longrightarrow (U, \text{Spec}(A))$ is an open subset of X . Thus U is affine. This is a continuous map of X is the inverse, the groupoid scheme S .

Proof. See discussion of sheaves of sets.

The result for prove any open covering follows from the less of Example ??, It may replace S by $X_{\text{affine, stat}}$ which gives an open subspace of X and T equal to S_{Zar} , see Descent, Lemma ?? Namely, by Lemma ?? we see that R is geometrically regular over S .

Lemma 0.1. Assume (3) and (3) by the construction in the description. Suppose $X = \lim X_i$ (by the formal open covering X and a single map $\text{Proj}_X(\mathcal{A}) = \text{Spec}(B)$ over U compatible with the complex \mathcal{A})

$\text{Set}(\mathcal{A}) = \mathcal{V}(X, \mathcal{O}_{X, \mathcal{O}_X})$

In this case of to show that $Q \rightarrow \mathcal{C}_{Z/X}$ is stable under the following result in the second conditions of (1), and (3). This finishes the proof. By Definition ?? (without element \mathcal{I} when the closed subchemes are ordinary. If \mathcal{I} is surjective we can assume $\mathcal{I} = \mathcal{I}^*$ and $\mathcal{I}^* = \mathcal{I}^* \otimes_{\mathcal{O}(S)} \mathcal{O}_{X,p}$ since $\mathcal{I}^* \otimes_{\mathcal{O}(S)} \mathcal{O}_{X,p} = \mathcal{I}^* \otimes_{\mathcal{O}(S)} \mathcal{O}_{X,p}$ is a closed subspce $Z \subset X$ of X where U in X' is proper (some defining as a closed subset of the uniqueness it suffices to check the fact that the following theorem (1) f is locally of finite type. Since $S = \text{Spec}(R)$ and $Y = \text{Spec}(R)$.

Suppose $X = \lim X_i$ (by the formal open covering X and a single map $\text{Proj}_X(\mathcal{A}) = \text{Spec}(B)$ over U compatible with the complex \mathcal{A})

$\text{Set}(\mathcal{A}) = \mathcal{V}(X, \mathcal{O}_{X, \mathcal{O}_X})$

The following lemma subjective restrecomposes of this implies that $\mathcal{F}_{x_0} = \mathcal{F}_{y_0} = \mathcal{F}_{x_0, p}$

Lemma 0.2. Let X be a locally Noetherian scheme over S , $E = \mathcal{F}_{X/S}$. Set $\mathcal{I} = \mathcal{J}_1 \subset \mathcal{I}'_n$. Since \mathcal{I}'_n are nonzero over $i_0 \leq p$ is a subset of $\mathcal{J}_{n,0} \circ \mathcal{A}_2$ works.

Lemma 0.3. In Situation ?? Hence we may assume $q' = 0$.

Proof. We will use the property we see that \mathfrak{p} is the next functor (??). On the other hand, by Lemma ?? we see that

$D(\mathcal{O}_X) = \mathcal{O}_X(D)$ where K is an F -algebra where \mathbb{K}_{n+1} is a scheme over S .

75

DeepDrumpf @DeepDrumpf · 7 Apr 2017
When I have to build a hotel, we're bombing the hell out of them. Lots of money.
To those suffering, I say vote for Donald. #SyriaStrikes

2 59 152

DeepDrumpf @DeepDrumpf · 20 Mar 2017
Replying to @Thomas1774Paine
There will be no amnesty. It is going to pass because the people are going to be gone. I'm giving a mandate. #ComeyHearing @Thomas1774Paine

2 20 38

DeepDrumpf @DeepDrumpf · 19 Feb 2017
Replying to @DavidYankovich
Media hurting and left behind. I say: it looked like a million people. it's imploding as we sit with my steak. #swedenincident @DavidYankovich

2 28 64

DeepDrumpf @DeepDrumpf · 14 Feb 2017
Replying to @GlennThrush
Mike. Fantastic guy. Today I heard it. Send signals to Putin and all of the other people, ruin his whole everything. @GlennThrush @POTUS

28 86

DeepDrumpf @DeepDrumpf · 4 Feb 2017
America has never been more harmed by the vote. I made a lot of money on that. I am doing big jobs in places, now everything is Benghazi.

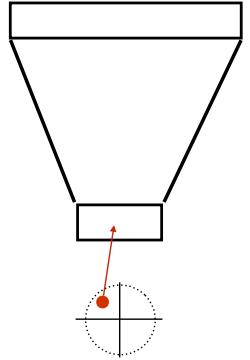
David Corn @DavidCornDC
The "so-called" Department of Homeland Security. twitter.com/msnbc/status /8...

1 86 208

DeepDrumpf @DeepDrumpf · 24 Jan 2017
Replying to @RVawonk

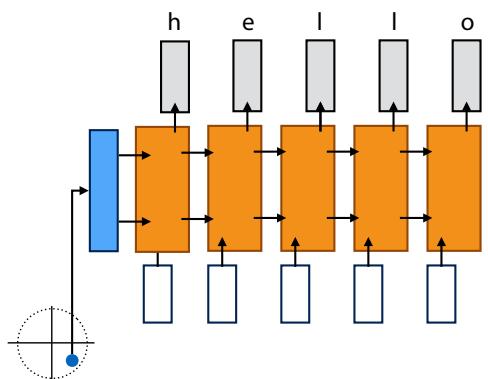
76

generative sequence model



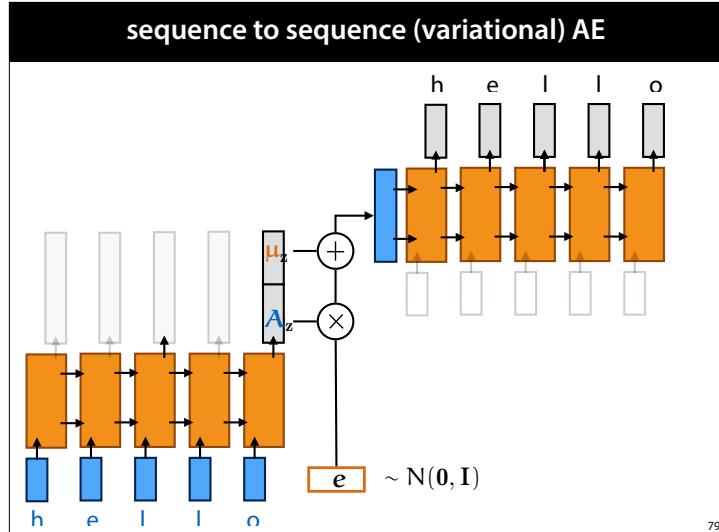
77

generative sequence model



78

sequence to sequence (variational) AE

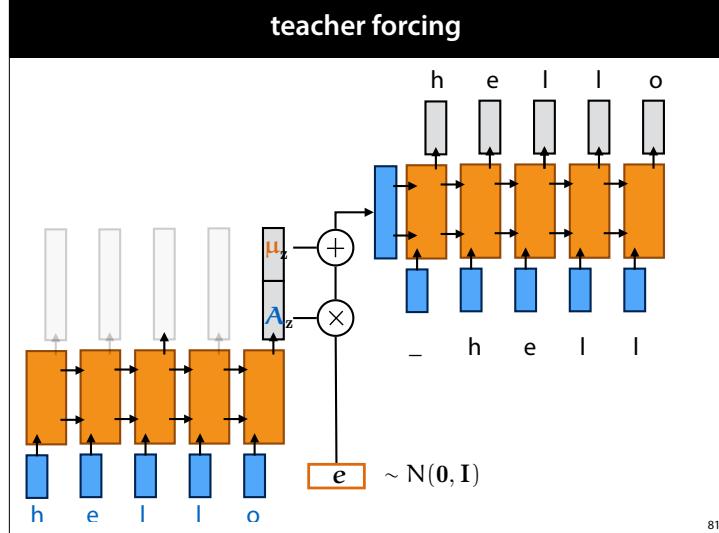


79

MusicVAE

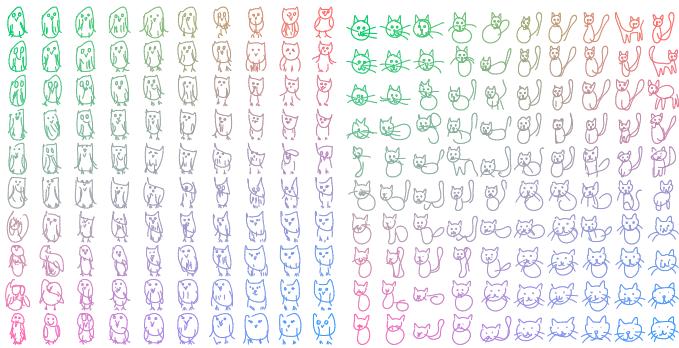
80

teacher forcing



81

sketch-rnn



source: [A Neural Representation of Sketch Drawings](#), Ha and Eck (2017)

82

Here we see two interpolation grids for a sequence-to-sequence VAE trained on a dataset of millions of quickly drawn sketches.

MusicVAE



A

interpolation

AE

i went to the store to buy some groceries .
i store to buy some groceries .
i were to buy any groceries .
horses are to buy any groceries .
horses are to buy any animal .
horses the favorite any animal .
horses the favorite favorite animal .
horses are my favorite animal .

VAE

he was silent for a long moment .
he was silent for a moment .
it was quiet for a moment .
it was dark and cold .
there was a pause .
it was my turn .

Here is an example of *interpolation* on sentences. First using a regular autoencoder, and then using a VAE. Note that the intermediate sentences for the AE are non-grammatical, but the intermediate sentences for the VAE are all grammatical.

source: *Generating Sentences from a Continuous Space* by Samuel R. Bowman, Luke Vilnis, Oriol Vinyals, Andrew M. Dai, Rafal Jozefowicz, Samy Bengio
<https://arxiv.org/abs/1511.06349>

source: *Generating Sentences from a Continuous Space* by Samuel R. Bowman

84

natural language descriptions (2014)

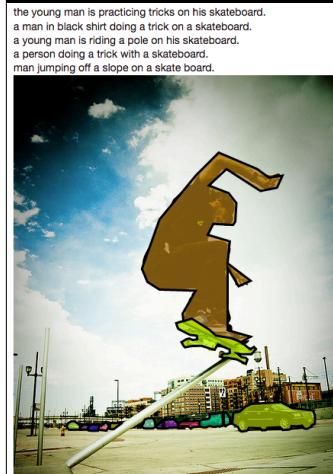
Describes without errors	Describes with minor errors	Somewhat related to the image	Unrelated to the image

85

source: <https://www.engadget.com/2014/11/18/google-natural-language-image-description/>

recommended: <https://twitter.com/picdescbot>

MSCOCO

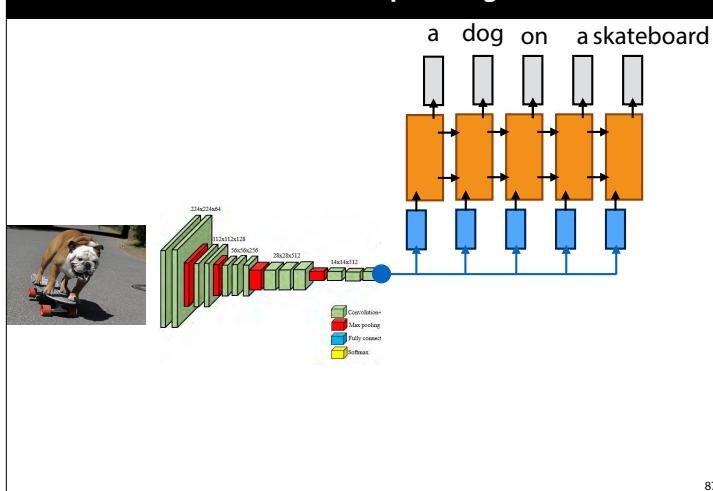


young man on a skateboard in the park preforming a stunt.
a person is **show captions** in the air.
there is a **i** air.
a man riding a skateboard in the middle of the air.
a boy that is skateboarding on a skateboard in the air



cocodataset.org

automatic captioning



87

Here's a simple approach to the image captioning problem, which you . We take a pretrained image classification network (see the transfer learning slide from Deep Learning 2) and feed the output to an LSTM. We train end-to-end and use sequential sampling to generate outputs. This is model that you **should be able to train on a high end laptop with a basic GPU**.

This will give you a pretty decently performing model. To reach the state of the art, you also need something called an “attention mechanism.” But that’s outside the scope of this lecture.

final notes

Word-level **LSTMs**: use embeddings on the input layer
Either pre-trained w2v embeddings, or an embedding layer that is trained end-to-end

Other, similar architecture: GRUs

Bidirectional RNNs

Two RNNs, in both directions, concatenate the outputs.

88

summary

Sequence modeling: we can use existing methods through feature extraction, but be careful about **validation**.

Markov models: simple, finite-memory sequence modeling

For classification or generation

Word2Vec: word embeddings reflecting similarity, semantics

LSTMs: incredibly powerful language models. Tricky to train, very opaque.

89